Citation

International Conference on APL >archive
Proceedings of the conference on APL '99 : On track to the 21st century: On track to the 21st century >toc
1999 , Scranton, Pennsylvania, United States

Choices in server-side programming: a comparative programming exercise

Authors
Robert G. Brown
Willi Hahn

Sponsors
NYSIGAPL : NY ACM SIGAPL Chapter
SIGAPL : ACM Special Interest Group on APL Programming Language
SIGPLAN : ACM Special Interest Group on Programming Languages

Publisher
ACM Press New York, NY, USA

Pages: 35 - 40 Series-Proceeding-Article
Year of Publication: 1999
ISBN:1-58113-126-7

http://doi.acm.org/10.1145/312627.312710 (Use this link to Bookmark this page)

ABSTRACT

One of the fastest growing and changing fields for software developers is in writing applications that are used across the "World Wide Web", which is in turn a client-server system that runs on the Internet. Servers, known by name, can be accessed over the Internet, and a protocol, known as HTTP (Hypertext Transfer Protocol) is used to send requests to servers for text, images (still and moving), audio, and other information. A very large amount of business will be conducted this way, now and in the future, having grown from almost nothing only 5 years ago. Application development on the Web is largely a matter of writing programs that are executed by the server, and we're going to focus our attention on two programming and operating environments that can be used to deliver server-side software.We would like to show how APL can be used to develop programs that run on the server, and we will

contrast it with another means of developing and delivering server-side programs. It is not our goal to deliver any judgment about whether or not one system is "better" than the other (we believe that any such statement would be simplistic to the point of being misleading); we intend only to share the results of our experience, help the reader understand what technologies that are available, and assist in making an informed choice if participation in a project of this nature is part of what the reader does.

CITINGS *2*

Dmitriy Zlobin , Alexei Roudometkine, Visual representation of document-oriented information on the web, ACM SIGAPL APL Quote Quad, v.31 n.3, p.247-254, March 2001

Vandana Pursnani, An introduction to Java servlet programming, Crossroads, v.8 n.2, p.3-7, Winter 2001

INDEX TERMS

Primary Classification:
D. Software
↳ **D.3** PROGRAMMING LANGUAGES
↳ **D.3.2** Language Classifications

↳ **Nouns:** APL

General Terms:
Languages

Peer to Peer - Readers of this Article have also read:

- We Talk to Everybody
  Linux Journal 2000, 74es
  Marjorie Richardson , Jason Schumaker , David Penn

- Editorial pointers
  Communications of the ACM 44, 9
  Diane Crawford

- News track
  Communications of the ACM 44, 9
  Robert Fox

- At the Forge
  Linux Journal 1998, 52es
  Reuven M. Lerner

- Forum
  Communications of the ACM 44, 9
  Diane Crawford

This Article has also been published in:

- ACM SIGAPL APL Quote Quad

Ch ices in S rv r-Sid Programming A Comparative Programming Ex rcis

Robert G. Brown and Willi Hahn

bob@acm. rg and whahn@fh-bingen.de

1. INTRODUCTION

One of the fastest growing and changing fields for software developers is in writing applications that are used across the "World Wide Web", which is in turn a client-server system that runs on the Internet. Servers, known by name, can be accessed over the Internet, and a protocol, known as HTTP (Hypertext Transfer Protocol) is used to send requests to servers for text, images (still and moving), audio, and other information. A very large amount of business will be conducted this way, now and in the future, having grown from almost nothing only 5 years ago. Application development on the Web is largely a matter of writing programs that are executed by the server, and we're going to focus our attention on two programming and operating environments that can be used to deliver server-side software.

We would like to show how APL can be used to develop programs that run on the server, and we will contrast it with another means of developing and delivering server-side programs. It is not our goal to deliver any judgment about whether or not one system is "better" than the other (we believe that any such statement would be simplistic to the point of being misleading); we intend only to share the results of our experience, help the reader understand what technologies that are available, and assist in making an informed choice if participation in a project of this nature is part of what the reader does.

2. AVAILABLE PROGRAMMING ENVIRONMENTS

2.1 Internet Server Auxiliary Processor (ISAP)

APL99, 07/99, Scranton, USA

ISAP is a product created by Lingo Allegro USA Inc. At this writing, it runs as an extension of Microsoft Internet Information Server (IIS) under Windows NT 4.0. Providing an "Internet Server Application Program Interface", IIS allows the design of IIS extensions, which become part of IIS; once they are loaded, extensions are executed in the same process space as IIS itself. ISAPI applications are loaded by IIS extensions and can remain in memory, removing response time and start up delays.

An essential part of ISAP is the use of Dyalog APL Version 8.1.3 or higher from Dyadic Systems Ltd., it uses a runtime version to answer nontrivial requests. Simple requests, such as server side includes, are handled by ISAP without using APL at all. ISAP is loaded and started by IIS when the first request for it arrives. ISAP itself starts Dyalog APL, which remains running in a wait state once it was started, which will provide for greatly improved response time for all subsequent requests. ISAP and IIS are multi-threaded in order to simultaneously processes several requests.

From the APL perspective, ISAP provides a high level interface to IIS. Traditionally, these interfaces are known as auxiliary processors (or APs), although the technique used here differs from the traditional design of AP's in that there are no shared variables, but internal and external functions connected to ISAP.dll. One may wonder how one workspace may manage multiple threads. ISAP is delivered with a workspace that includes an application manager. It takes care of multiple requests that need the execution of the same or different functions at one time. Dyalog APL's namespace technique and the ability to clone namespaces when needed are exploited heavily and efficiently. At this writing a new feature of Dyalog APL, multi-threading (implemented via the & operator) is not exploited by this architecture. It would be logical to expect real performance and response gains when & is implemented with OS-native threads.

The current ISAP version (5.1) differs from previous versions in that a set of script tags is now supported, reducing or in some cases eliminating the need for APL programming. It is also possible to define new tags and support them with APL functions. Therefore, ISAP is not only a tool to write server side scripts using prescribed tags, but also it is an extendable tool allowing the definition of an extended HTML script language. Lingo proposes to call the file extension for these ISAP scripts ".xml" due to that feature. "XML" stands

for eXtendable Markup Language. It is possible to register different extensions for IIS instead of ".xml" for script files. This can be used to run several copies of ISAP in parallel answering to different script files.

Whenever IIS receives a request for a document with a registered extension, it passes that request to ISAP, which parses it for tags and starts the appropriate actions. The same is true when IIS receives a request that explicitly refers to ISAP.dll. For example an ISAP starting request could be:

`http://hostname.domain[:port]/cgi-bin/isap.dll?ws=sample`

or

`http://hostname.domain[:port]/isapdocs/sample.xml`

The first one assumes that the namespace "sample" would contain a latent expression (in the application manager framework) named "_COM" which, for example, is a nested vector of commands to execute in order to produce the HTTP answer to the request.

In the second case, the document "sample.xml" serves as a template HTML file with or without ISAP tags included. By parsing these tags, ISAP executes appropriate functions depending on the tags. Therefore, it is not necessary that APL functions deal with large volumes of data. The static parts of the document remain in the ISAP channel and only parts of the document are generated by APL functions, a technique faster than producing a whole document with APL software.

2.2 CGI – The Common Gateway Interface

CGI is a specification that provides some standards regarding the creation and use of software on a server (these days, usually an HTTP or Web Server). and how information is passed between clients (a web browser is a good example of a client) and the server. Programs are started by the server at the initiation of users (clients), who request that a given program be run by sending the URL of the program, and parameters needed to run the program.

Usually, this request is part of the HTML that is retrieved from the server by a prior request from the client. We thus have something of a dialogue:

Client: Requests HTML page

Server: Supplies it.

Client: Transmits input data and CGI program name (URL)

Server: Runs the requested program, passing parameters to it, and returns a properly formed result to the client.

The result of running a CGI program is usually HTML, which is used to call a program via CGI, so the client and server dialogue can be extended in many useful ways, with essentially infinite variability. The result is a sequence of interchanges which appears very much like an interactive session between user and one or more programs on the server.

Now, CGI is very graceful, in that it is in essence language-independent, and only specifies the behavior of the components (including the server-side program you wish to call), and the accepted formats for data exchange.

Implementation has been another matter. Early CGI-capable web servers would load the program being called into memory each time a user called them, which degrades perf rmance and response time. Since some of the languages that are used to write CGI programs require their own interpreters or run-tim libraries, these would have to be loaded and initialized before the CGI program could run, degrading per-formance further. As more web servers were put in service, and the frequency of transactions on some web sites climbed, these aspects of the CGI implementation placed very real limits on both response and performance.

Several changes have been made, not to the CGI standard, but to the web servers themselves. One very popular server, known as "APACHE", can be configured so that once a CGI program is run, it remains in memory, along with any libraries or other support software. Subsequent calls to that program will not require any disk access, improving response and maintaining good performance. Another improvement to Apache is that whole interpreters can be loaded into the web server at startup, so those essential components are always in memory, again improving response time. At this writing, Apache modules exist to load PERL and PHP interpreter/compiler packages at startup, and there are plans to do the same with an APL interpreter in the near future.

3. THE BOOKSTORE APPLICATION

In order to conduct our comparison, we needed a non-trivial example, something that was not out of the ordinary for web-based applications that are quite common. However, we also didn't want to write something so large and complex that quantifying our experience would itself become a daunting task. We were relative novices, and starting reasonably small seemed to be an appropriate choice. Since "E-commerce" is spoken of as being a justification for many web sites, we decided to write server-side software that would drive an order form for (since we are both teachers of APL) books about APL, mostly left over from ACM inventory reduction and the old APL Press. Not only does each item carry a price, but there are a few "special packages" that are offered, a discount on pairs of books when purchased together, for example. We wanted to handle all the arithmetic related to this, prompting for a shipping address, and handling any errors that would arise during these user interactions.

Both of the authors are APL programmers with a fairly large amount of experience, one of us has alsb spent a fair amount of time writing C and C++ applications, and that person had the additional assignments of installing and configuring a web browser (APACHE) and learning one of the most common languages used in CGI: PERL. The other of us had to learn how to use Dyalog APL, and con-figure Microsoft Internet Information Server (IIS) and ISAP. We kept track of our time for all of these tasks, primarily so w could isolate the amount of time we actually spent writing the application.

We also decided to introduce one simplification: The system would not even attempt to calculate shipping charges for the books being purchased. We understand that a real-

world, high-volume application would probably have to perform this function (as well as take credit card orders, perhaps track the progress of an order, and so forth), but we didn't want to create an application of quite that scale, and we felt that most of what we need to learn that was an intrinsic part of server-side programming was present in our definition of the system. An order is submitted when a mail message, which is essentially a completed order form with the destination address, is sent. Weight and shipping computations are done by a mailing service, and the transaction is completed via email, a message being sent to the customer and to the person who needs to pull the ordered items from inventory.

3.1 The bookstore solution with Dyalog APL and ISAP

3.1.1 Approach

There are several choices to solve the stated problem, depending on the desired appearance of the pages and the comfort of the user during the ordering process. The maintenance of the software is also important, as the stock of books, papers and pamphlets isn't static at all. Prices could be changed as well as the rebating policy.

It is important to maintain the information that was keyed by the user. That means if a value is entered but the order failed due to some mistake, or it was not a final order at all but just an evaluation request, then the user should be saved the work of rekeying everything again.

The ISAP script solution for this problem consists of the following steps. First a variable, in this case, a character variable is declared for every input field somewhere in the .xml document

```
<!--#ISAP CAR=B1 NAME="b1" DEFAULT=""-->
```

This causes ISAP to define an APL variable B1 that is associated with a corresponding CGI variable with the name b1 (this can be the same name, without confusion) belonging to an HTML input field of whatever type. Its default value here is empty as long as the CGI variable has not been assigned a value by entering data in a field or pressing a button. The first time the user gets the HTML document, he or she sees empty fields.

Next, the action to be performed by submitting data to IIS is declared to be the same script in which the form with all the input fields resides. Pressing the submit button in the form causes ISAP to interpret the same document again, now with data filled in. The corresponding script lines are:

```
<form
method="post"action="books.xml">
```

where "books.xml" is the document containing this form, and:

```
<input name "b1" type="text"
value="<!--#ISAP EXEC="B1"-->">
```

This line uses an ISAP Execution tag in the value field. Thus, the value of B1 is included in the answer, since B1 is a variable associated with the CGI variable b1, the returned value now is what was sent with the CGI variable b1 during the previous request. The user should get back what was originally sent. With appropriate page layout and positioning, the user may see a rather static picture on the screen.

That has to be done with all input fields the same way with different names, which may be meaningful for some fields. For example FNAME and LNAME or STREET for address related input fields is a good choice that eases maintenance of the corresponding APL programs. So far, there was no APL programming at all involved.

A second goal in designing the prototype was maintainability. It should be possible to add or remove books, papers or pamphlets without recoding the APL calculation functions.

In order to achieve this, some hidden input fields surrounding the books, papers and pamphlets entries were introduced, without association of ISAP variables to them. They served just as markers in a matrix which ISAP copies to the namespace. Three pairs of start and end markers were introduced: sbooks, ebooks, spapers, epapers and spamps, epamps, which marked the beginning and the end of the corresponding order fields for books, papers and pamphlets. ISAP provides a matrix in the application manager workspace called _IN that holds the names of all input fields in the first column in the order they occur in the form and the values entered in the second column.

It is a rather easy task in APL to compute the positions of the "start" and "end" fields for the three categories: books, papers and pamphlets, as well as which and what quantities were ordered. This is possible without referring to the individual titles. If some items are to be added or deleted from the list, this may be done by simply adding or deleting entries in the file "books.xml" and updating the price and rebate information for the APL function e.g. in a global variable. However, no recoding would be necessary.

When the user presses the button named "Preview" the following ISAP tags become essential. Of course, it is interpreted each time the user asks for this document.

```
<!--#ISAP WS=bbs-->
```

This tag tells the application manager which namespace to clone and examine for a "latent expression". Only the first detected WS-tag in a document is interpreted. Here it is "bbs". We have no latent expression in the bookstore application. So what happens? There is an EXEC tag that explicitly indicates what to do: execute the APL expression "store" which in our case is a function, which simply looks

whether the "Preview" button has been pressed by inspecting the variable SUBM1. If it is mpty, the button had not yet been pressed and it returns an empty character vector that replaces the EXEC tag in the HTML document.

```
<input type="submit" name="subm1"
value="Preview">

<!--#ISAP CVAR=SUBM1 NAME="subm1"
DEFAULT=""-->
<!--#ISAP EXEC="store"-->
```

As this tag is executed in each request, the APL function "store" branches depending on the button information it has.

If the variable SUBM1 is not empty, the function "store" calls the function "calculate". This function analyses the user input and returns either a character matrix with all the necessary order information or an error message, when for example no orders at all are given or invalid data is in one of the input fields. This error message replaces an ISAP tag in the template file.

With nice formatting, a field with a different background color popup, presenting an appropriate error message.

If an order is entered successfully, another table shows what the user is going to order and a second button named "ORDER NOW", initially disabled, is now enabled.

Disabling and enabling the button is also done by a tag; the line looks like:

```
<input type="submit" name="subm2"
value="Order now" <!--#ISAP
EXEC="ACTIVE"-->>
```

Of course, a character variable "ACTIVE" was declared before. Moreover, the current value of ACTIVE is passed from and to APL via a hidden input field. At start time, its default value is "disabled", and it becomes enabled as soon as the user has successfully done a "preview" request. It becomes "disabled" again when the user should happen to "mistype" the order form.

As HTTP is a sessionless protocol, it is essential to know what data belongs to which user. There are several possibilities supported by ISAP, a common one is to use "cookies", or short strings of data, stored on the users' computer.. Another would be to create an identity before any dialog starts. Some professional sites do this, asking in advance for credit card information, a policy saves time when these sites are visited repeatedly by the same users.

Here another simple solution was implemented. As the amount of data isn't big at all, the order information belonging to one certain user is passed back and forth once it is created using a hidden input field together with an ISAP character variable.

The main information is the character matrix containing a simple invoice. It is used for email and slightly formatted as HTML table for the display on the order page. Together with the above mentioned technique to use the same file "books.xml" as both, a template and a script, the user gets the impression that the dial g back and forth takes place on the same page. Fields are just opened on demand and widened or shrunk as is appropriate.

The complete script "books.xml" and the APL namespace may be downloaded via anonymous ftp from

ftp://watchfire.dhis.org/pub/apl99/isapbooks,

and other sites as contacting the authors will reveal.

3.1.2 Quantitative Results

This development was done while learning every aspect of the environment: Managing IIS, installing ISAP, modifying the ISAP installation, creating virtual directories, and testing the different tags. This work was done after regular work, during night sessions, meaning the following results should be taken with care. They are influenced by those circumstances.

The total amount of work of approximately 20 hours turned out to be consumed mostly by testing, stopping and starting the www services on the NT server. The HTML-XML scripting could be done rather quickly by copy and paste operations with a simple editor in about 6 hours. The APL programming, once the mechanisms became clear, was about 4 hours. The remaining 10 hours go on the following accounts: lack of experience, overseen restrictions in ISAP, poor error messaging of IIS, ISAP and Dyalog APL. "Could not open the requested page", "The server returned an unrecognized response" or "1 File not found" are some of them. Reading such a message caused probably by a misspelled noun is not helpful in trying to find the misspelling. Lines of code including comment lines in APL were without any attempt to reduce them about 158 lines. Counting only the problem-related lines without the utilities comes to 53 lines.

3.2 The CGI Approach

Prior to this effort, and in conjunction with other projects, a number of e-commerce sites were examined, for usability, style, and the visible portions of methods used by the developers of these sites. There are also a couple of characteristics of HTTP that introduce some interesting problems, and dealing with them would be another aspect of creating the desired interaction.

HTTP does not really encompass the idea of a "session", or series of transactions that are considered to be a unique and distinct dialogue between a particular client and the server. Therefore, some sort of connection has to be made by another means, and there are several ways to do this. For example, the server could assign a number, or other identifying information to each request at the start of a dialogue, and this could them be embedded in subsequent HTML passed between client and server. Another method would be to transmit a "cookie", or small file that holds information about the transaction, surely enough to identify users in a table on the server. Another way to perform session-like continuity is

not to identify users at all, but simply embed any necessary information into each HTML interaction, so that processing can resume even after long lapses in communication between client and server. This approach is common for smaller, less complex transaction chains (as wee see in this application), and permitted the development and testing of code that didn't rely on a lot of side effects, and could be easily tested.

A couple of shortcuts were taken, and the applications would be implemented differently with a more varied (and variable) inventory. Some interesting discoveries helped as well; it became clear that HTML input field names could be almost anything, and a fair amount of data (what would normally be key field values) was encoded into the field names. In a larger scale application, these become keys in a database. Here, the choice was to embed the data items in the HTML itself, in the CGI field name in the form "type;price class;title". This is not too secure, but with manual intervention later on in the ordering process, a fairly small and static (but shrinking, we hope!) inventory, and considering that this was in essence a prototype, there was no conceptual difficulty introduced by these assumptions. Inspection of a number of e-commerce sites on the Internet shows that they are more complex as demanded by the perceived need for diversity, complexity, and security.

The nature of the dialogue was to present a form, and allow the user to enter quantities. Pressing a button at the end of the form would then compute the price of the ordered books (including discounts), and display a total. If the user realized that a mistake had been made, this was a chance to go back to the original form and make corrections. To complete the order, the user would then fill in a form with name, shipping address, and email address. Pressing a button on this second page would submit the order. There are also a number of other pages (error messages) that the user might see if impossible or incomplete data is entered.

The first page is static HTML, complete with a call to a CGI program that runs on the server when the button is pressed. The actual processing starts when a button is pressed, and there's an HTML fragment that calls the CGI program that looks like this:

```
<FORM METHOD=POST ACTION="/cgi-
bin/bookform.cgi">
```

The program "bookform.cgi" is written in PERL, and it received the quantities of each item desired by the user, checks to see that positive integers have been entered, and computes the order costs (per item, any discounts, and the total). Then an HTML page is generated that shows that information, with input for shipping information. What the user does not see is that the order quantities are kept in this dynamically generated HTML page, in hidden input fields, so that they will be sent back to the server when a button is pressed. This dynamic page has an embedded call to a second CGI program.

When the second page is processed, the new input fields are checked to make sure that input looks reasonably complete, the hidden input fields are used to generate the body of the invoice/mail message, and the customers name and address information is formatted at the top. An email interface is then invoked to send this information as an email message to the two destinations. Finally, a "Thank You" HTML page is sent to the HTTP client, so the customer is told that the order has been accepted.

3.2.1 Quantitative Commentary

A lot of this code was relatively crude, but as this is true of many initial development efforts; a good deal was absorbed about this type of programming and run-time environment. When this is rewritten, many of the basic housekeeping components of PERL will be used. For example, a PERL script would be created to periodically (on demand, or when changing the collection of books being offered) regenerate the order form, and the system of item keys and a server-side file for product identification would be developed. Some better programming practices would be employed, such as more subroutines and less in-line HTML (many of the error messages are in-line PERL "print" statements, in their own subroutines); a lot of static HTML fragments can best be put on files, and simply "printed" into place at run-time. The amount of information embedded in the second form would be reduced to a single order number, which would be assigned while processing the first form. Now that PERL is fairly clearly understood, some time would be spent evaluating the many scripting/tagging languages and packages available in this environment, reducing overall complexity while placing greater reliance of more advanced practices, where possible. Finally, several people have pointed out that it would be natural to tie this to an inventory table, reducing item counts as orders came in. Again, that would be a good "next version" to see how this technology would scale up to the next level of complexity.

3.3.2 Quantitative Results

There were approximately 40 hours of reading and experimentation required to learn PERL so that coding would move forward with confidence, knowing that what was written would behave as intended (or close). Coding of the application itself took about 15 hours, with 10 additional hours of testing (Each block or function was tested as it was written, and then tested it in context once it was finished). Once the application was working, there were another couple of hours doing cosmetic work on the HTML, so it was pleasing in appearance. Some of this can be a bit tedious; 4 different web browsers were used to make sure that the page looked correct in each browser; the same treatment is recommended for any serious web site development effort.

Testing itself was rather straightforward, by substituting one routine for another, the input that is expected from the web browser can be taken from the command line, and a new PERL code fragment can be tested from the OS command line much the way one might test an APL function, naming the script and stringing the arguments along. Output will be HTML, but comes back to the command shell so it can be seen, or can be redirected to a file. For proof positive of what

the result looks like (although by this point you will be fairly fluent in HTML; the ability to look at HTML and visualize the page is possible, something of an acquired talent), this can then be read by most browsers.

Having climbed this learning curve, this type of application can be written in about 50-65% of the time taken here. In round numbers, this means that it will take between 12 and 16 hours to write the next version, which seems reasonable. Even with significant extensions to the architecture, as may be required to integrate databases in an extended design, only a few additional hours would be required for the next version.

The server-side PERL code is 279 lines of code, but let's break this down a bit more. If we remove all the lines that separate functions, or have a single "}" character (PERL is a block structured language, and every block, even if it has only one line, requires braces), the count drops to 179. There are 50 lines of completely static HTML in-line, and about 15 lines of PERL that are equivalent to the ISAP application manager passing data from the interface to the application code, in each PERL script. Of the remaining 102 lines of code, 15 are declarations or simple expressions that can be stored as global variables in APL, or are function headers. The remaining 85 lines comprise the logic, non-static HTML generation, error-checking, and pricing. The pricing code itself is only 11 statements long, and it is all written in PERL primitives.

The PERL scripts named here, as well as the static HTML may be downloaded via anonymous ftp from

`ftp://watchfire.dhis.org/pub/apl99/cgibooks` and other sites as contacting the authors will reveal.

3.4 Comparison of our Experience

3.4.1 Advantages and disadvantages of APL

The calculations could be done in an array oriented manner without any explicit reference to the input fields. The fairly simple rebating rules could be implemented using Boolean array functions painlessly; there are no loops. The main script interpreting function "store" consists of 26 lines due to the fact that control structures are used. Without them it would be much more compact, but unreadable. A major disadvantage of the APL approach is that it is tightly coupled to IIS, Windows NT, and Dyalog APL. At this writing there is no way to port the solution to a different OS, different APL interpreter, or a different Web server.

3.4.2 Advantages of Using CGI and PERL

Perhaps the biggest advantage of using PERL, FAST-CGI and related technology is that there is very broad support for it. This PERL/CGI application can be ported to a large number of platforms, from NT-based servers to AS-400's, many UNIX platforms, even mainframes, without modification. The platform on which this software was developed (Linux and Apache) is the fastest growing operating system/web server combination in the computer industry; assured of broad support for years to come. The wide acceptance of PERL and CGI, as well as the Open Source culture in which PERL was developed, means that there are thousands of web sites out there running on this platform, a robust collection of tools for server-side software construction, and many developers openly willing to share software for a large number of solutions.

There are many, many packages available (including tag-based language extensions) for the CGI environment and PERL programmer, which provide a great deal of added value and utility to anyone working in this environment. A lot of innovation in web servers (such as server-side includes) have been developed in the Open Source development efforts surrounding the expansion of activity on the world wide web.

PERL itself seems to be somewhat more array-oriented than most of the block-structured languages. There are "associative arrays", which allow the programmer to index values by character string, and several operations that refer explicitly to the shape of arrays, reordering arrays, and implicit nesting of character arrays. Some operations are defined on whole arrays, which as we can appreciate, speeds up a number of steps. One of the reasons PERL coding ability was acquired so quickly was that there are several important and strong similarities between PERL and APL; despite time spent as a C/C++ programmer, array-oriented techniques run deep.

4. CONCLUSION

The joint conclusion is that we have shown (and learned) a good deal about how these technologies support server-side processing. For applications of this level of complexity, it would appear that both environments are about equally strong. ISAP is currently evolving into being more than a simple interface between IIS and Dyalog APL; this makes comparison with code written purely in native code a bit harder. We know that there would be more APL code without ISAP tags, and it's reasonable to think there would be somewhat less PERL if we used scripting tags or special libraries in that environment. Therefore, for applications of this complexity, we believe these results to be similar.

It is difficult to say how the economics change were we to move on to larger and more complex applications. We do have some suspicions, however. APL will be more suitable as the scripting language around ISAP becomes more robust and capable of handling more of the housekeeping tasks associated with constructing HTML, like many other sets of specialized scripting tags. More complex, rapidly changing and array oriented applications are where APL shines, and we would expect it to shine there as well, right across the Internet.

Citation

Conference on Object Oriented Programming Systems Languages and Applications >archive
Addendum to the 1997 ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (Addendum) >toc
1997 , Atlanta, Georgia, United States

Experiences using object data management in the real world

Author
Akmal B. Chaudhri

Sponsor
SIGPLAN : ACM Special Interest Group on Programming Languages

Publisher
ACM Press New York, NY, USA

Pages: 144 - 149 Series-Proceeding-Article
Year of Publication: 1997
ISBN:1-58113-037-6

doi http://doi.acm.org/10.1145/274567.274606 (Use this link to Bookmark this page)

> full text > references > index terms > peer to peer

> Discuss > Similar > Review this Article > Save to Binder

> BibTex Format

↑ FULL TEXT: Access Rules

pdf 736 KB

↑ REFERENCES

Note: OCR errors may be found in this Reference List extracted from the full text article. ACM has opted to expose the complete List rather than only correct and linked references.

Loomi98 Akmal B. Chaudhri , Mary Loomis, Object databases in practice, Prentice-Hall, Inc., Upper Saddle River, NJ, 1998

Zorn95 Benjamin G. Zorn , Akmal B. Chaudhri, Object database behavior, benchmarks, and performance: workshop addendum, ACM SIGPLAN OOPS Messenger, v.6 n.4, p.159-163, Oct 1, 1995

↑ INDEX TERMS

Primary Classification:
D. Software
↳ **D.1** PROGRAMMING TECHNIQUES

Additional Classification:
D. Software
↳ **D.2** SOFTWARE ENGINEERING
↳ **D.3** PROGRAMMING LANGUAGES
↳ **D.3.2** Language Classifications

↳ **Nouns:** C++

H. Information Systems
↳ **H.2** DATABASE MANAGEMENT
↳ **H.2.4** Systems
↳ **Subjects:** Object-oriented databases

I. Computing Methodologies
↳ **I.7** DOCUMENT AND TEXT PROCESSING
↳ **I.7.2** Document Preparation

↳ **Nouns:** SGML

General Terms:
Design, Languages, Standardization

↑ Peer to Peer - Readers of this Article have also read:

- We Talk to Everybody
  Linux Journal 2000, 74es
  Marjorie Richardson , Jason Schumaker , David Penn

- Editorial pointers
  Communications of the ACM 44, 9
  Diane Crawford

- News track
  Communications of the ACM 44, 9
  Robert Fox

- Forum
  Communications of the ACM 44, 9
  Diane Crawford

- At the Forge
  Linux Journal 1998, 52es
  Reuven M. Lerner

Experiences Using Object Data Management In the Real World

Akmal B. Chaudhri
akmal@soi.city.ac.uk

Abstract

The OOPSLA '97 Workshop on *Experiences Using Object Data Management in the Real-World* was held at the Cobb Galleria Centre in Atlanta, Georgia on Monday 6 October 1997. This report summarises some of the commercial case-study presentations made by workshop participants.

Introduction

This workshop was organised in an attempt to bring together academics, users and vendors from the object database community, following the previous successful workshop on *Object Database Behavior, Benchmarks, and Performance* at OOPSLA '95 [Zorn95]. The workshop Call for Papers (CFP) attracted a mixture of submissions from academics and users and 16 position papers were selected for presentation. There were several main themes to the submitted papers:

- Object Database Applications and Design.
- Benchmarks and Standards.
- Architectures and Frameworks.
- Object to Relational Mapping.

The position papers were organised into short and long presentations, with an opportunity for workshop participants to pose questions and raise issues for discussion after each presentation.

The remainder of this paper is organised as follows. The next section describes some of the commercial case-study presentations in the order that they were presented, (due to space constraints, however, other workshop presentations cannot be summarised in this paper). This is followed by the major conclusions.

In the Trenches with ObjectStore

This paper was written by David Hansen, Daniel Adams and Deborah Gracio and presented at the workshop by David Hansen. All three authors are with the Information Systems Department, Pacific Northwest National Laboratory (PNNL).

David's presentation focused on experiences that his group has had using ObjectStore for a Scientific Data Management (SDM) system.

Fast performance is provided by ObjectStore through its Virtual Memory Mapping Architecture (VMMA), but at the expense of pointer safety and increased burdens on the programmer. David concluded that safe pointers, at the expense of a little overhead, would have been better to avoid some of the design compromises that needed to be made for SDM.

The second issue discussed by David was that of physical database organisation. In ObjectStore, objects that are likely to be referenced together can be placed in clusters (fixed-size containers) which reside in segments (expandable storage containers). Cluster size can only be set at creation-time. Similarly, segments need to be carefully sized. David argued that such physical considerations should be the responsibility of a Database Administrator (DBA), rather than hard-coding them directly into each application.

The third issue raised by David was that of vendor marketing. He (very diplomatically) suggested that the marketing department at Object Design, Inc. (ODI) was ahead of product capabilities. For example, SDM uses the Silicon Graphics platform, running the IRIX operating system. However, ODI does not appear to give high priority to this platform in terms of support, development or maintenance.

The fourth issue relates to tools support. David described some experiences with ODIs Internet Solution Suite (ISS). He argued that several tools of interest for SDM, such as ObjectForms and the Extended Object Manager were not up to quality. For example, the SDM team wanted to view objects from their database, but found ObjectForms to be very limited in capability and only providing relational-like data access. Similarly, the wrapper class for the Verity Developer Kit (VPI) was somewhat limited in capabilities. Again, the application programmer was left to deal with performance and safety problems.

David concluded that experiences with ObjectStore had been very mixed, but on the whole, the product was a superior solution to using relational or object-relational products for SDM.

CORBA-Based Applicati ns, Services, and Object Data Management

This paper was written and presented at the workshop by John Chen. The author is with Silicon Graphics, Inc.

John described the results of a project to build a set of CORBA applications and some object services needed by those applications. One of these services being persistence. A pure object database was used as the first persistence mechanism and John's experiences reported below are based on this.

One of the first issues discussed by John was how to locate a database object for a particular CORBA object:

1. Use the OID generated by the object database.
2. Use the names selected by the CORBA object.
3. Use the names managed by the Persistence Service.
4. Use the OID generated by the object database and add a tag generated by the Persistence Service.

The first three options are unsatisfactory, since the particular object database system reuses OIDs (option 1), which makes it difficult to keep track of deleted objects or a mapping table is needed for names → OIDs (options 2 and 3), which introduces an overhead. The fourth option does not incur a mapping overhead and was the one selected.

In a distributed environment, it is important to access local data as much as possible to avoid unnecessary network traffic. The CORBA applications do this by using the *Federated Database* mechanism of the object database. This enables those objects that are owned by a particular CORBA server to be physically held on the same machine as that server. Furthermore, good concurrency is achieved by using the *Container* mechanism of the object database - all objects of the same type as their CORBA objects are held in the same container. Also, database interaction time was found to be the limiting factor for I/O-bound applications, so tradeoffs were necessary to achieve good throughput, such as batching requests and committing transactions per request batch.

John then discussed several issues related to database management and application packaging. These included shutting down the database server in a distributed environment - not an easy task, since all applications need to shut-down first and this can only be done when any outstanding transactions either commit or abort. A related issue is that of the lock server used by the particular object database which also requires careful handling. The object database also provides very minimal security and access control, so user authentication and authorisation are performed at th CORBA object level. Schema managem nt was also a little difficult in this object database system, since all application schemas are held in one database.

Overall, despite the problems just described integrating CORBA and the particular object database, John ended on a positive note and felt that the inconveniences and problems were not insurmountable.

Using Objectivity on the IRIDIUM[1] System

This paper was written and presented at the workshop by Jeff Garland and Dick Anthony. Both authors are with Motorola, Inc.

The IRIDIUM project involves a large number of components, including satellites in low earth orbit, ground stations, etc. OO software and development techniques are currently being used to develop the System Control Segment (SCS) which is central to the entire IRIDIUM system.

Objectivity/DB has been successfully used for sometime in several SCS ground sub-systems. Other sub-systems also use OO tools and techniques, such as Orbix, Rogue Wave and ILOG. An obvious issue is that of combining these diverse tools to work successfully and this was achieved in IRIDIUM by the SCS team developing high-level design and integration techniques, including the use of design patterns. For example, one use of Orbix within SCS is to perform distributed event notification to database clients. This is modelled by a design pattern, called `Observer`. When a database client receives an alert, it requests the database object using Objectivity/DB's direct interface.

One of the problems with any large-scale development is trying to ensure good software practices. Within SCS, this was addressed by a series of measures, which can be summarised as follows. First, coding guidelines were used with Objectivity/DB. In particular, calling `ooUpdate` for all non-const methods and ensuring that there were no direct interfaces across major sub-systems. Second, by a framework of objects to unify Objectivity/DB's features. These included enhancements to session classes, using wrappers

[1] IRIDIUM is a registered trademark and service mark of Iridium LLC.

around transactions, the development of specialised collection classes (since the collection facilities provided by Objectivity/DB were inadequate for this project) and developing an error handler to convert Objectivity/DB errors into exceptions. Third, standard make rules were used to ensure that all system-level schemas were correctly built. Fourth, C++ coding guidelines were used and enforced through code inspections. Fifth, all system-level interfaces were carefully developed with Programmer Interface Guide.

Jeff and Dick summarised their presentation by noting that Objectivity/DB was a key element of the ground system software and was being successfully integrated and used with other technologies through careful attention to interfaces, implementation issues and using only the core features of the product. The SCS team had also developed a framework for the rapid development of applications with the object database, which reduced the development time for new applications. Finally, an important factor to the success of the overall project was the large number of experienced developers being used.

The Electronic Library Project: SGML Document Management System Based on ODBMS

This paper was written by Philippe Futtersack, Christophe Espert and Didier Bolf and presented at the workshop by Philippe Futtersack. The first two authors are with Direction Etudes et Recherches, Electricité De France (EDF) and the third author is with Ingenia SA.

At the R&D division of EDF, large quantities of documents are generated. These include such things as technical reports, project proposals, organisation charts, etc. These contain large quantities of text, image and possibly video data and currently number 140,000 documents. There are also hyperlinks between documents, to represent relationships, which are stored in separate HyperIndex documents. At present there are 80,000 SGML documents and 60,000 HyperIndex documents. Since the actual documents represent a considerable corporate asset for EDF, one aim of the project described by Philippe was to make this information available in a more accessible and consistent manner.

The project team decided to use SGML, since there is already an ISO standard and due to its widespread use in many other industries. To enable any links to be defined between any documents an extension, called HyTime, was also used. Documents stored in the object database are represented as a tree structure with an average of 1,000 nodes and leaves. Each node or leaf is an instance of the ElementSGML class with the HyTime links being generated by a HyTime engine developed by the team. The actual database schema comprises more than 70 C++ classes and can be used to store any SGML, XML or HTML document. O_2 was used to provide persistence and was chosen because of previous experience some members of the team had using this product.

Since the full-text query facilities provided by O_2 were inadequate for this project, the development team decided to use the Search '97 full-text engine from Verity. The indexing was performed on the textual content and structure of the 80,000 SGML documents by scanning each document tree in the object database. A web interface provides the mechanism for end-users to query the database with the results being displayed in a table showing the documents that satisfy the selection predicate. If the user selects a particular document for display, this requires a recursive tree traversal of 1,000 objects on average. For a document search or displaying a document, the response time was found to be less than 1 second.

Philippe concluded that the integration of three different technologies, namely an object database, a full-text retrieval engine and a web interface, was far better than expected and provided superb performance. The object database also demonstrated that it was very efficient and well-suited for th persistence mechanism for a structured document management system.

O_2 and the ODMG Standard: Do They Match?

This paper was written and presented at the workshop by Suad Alagic. The author is with the Department of Computer Science, Wichita State University.

Undoubtedly, one of the major reasons for the slow uptake of object databases for commercial applications has been the lack of appropriate standards. The standards work that was begun a few years ago by the Object Database Management Group (ODMG) was forced to work on standards issues after many object database vendors had already begun offering commercial products. Since object databases provide seamless bindings for object-oriented programming languages, eliminating the so-called "impedance mismatch" problem, the task of standards development has been further complicated as each object-oriented language has its own data model and type system. Suad's presentation, therefore, focused on three issues: (i) the type system, (ii) the model of persistence and (iii) non-traditional object-oriented features.

For the type system, the ODMG Object Model attempts to provide a common model for several object-oriented languages: Smalltalk, C++ and Java™. However, as Suad noted, Smalltalk is dynamically typed whilst C++ and Java are mostly statically typed. Furthermore, there are currently large differences between the typing systems of C++ and Java. This affects parametric polymorphism and obviously has implications for the correct typing of collections which are essential for querying. For example, the ODMG Object Model has a root object type that is supported by the Smalltalk and Java bindings, but not the C++ binding. Similarly, the C++ binding uses templates to support parametric polymorphism, whilst the Java binding does not.

When looking at a particular object database, such as O_2, Suad highlighted a number of significant problems. For example, O_2 supports the Object Query Language (OQL) and C++ binding as defined in the ODMG standard, but its Object Definition Language (ODL) is totally different from the ODL defined by ODMG and differs from the type systems of Smalltalk, C++ and Java. The O_2 object model is therefore different from the ODMG Object Model. Furthermore, whilst parametric collection classes are supported in the C++ interface of O_2, as required in the ODMG standard, they are not supported in the actual O_2 system.

For the model of persistence, Suad showed why orthogonal persistence is desirable for object databases and where the ODMG standard and O_2 are again controversial. Complex object support, he argued, cannot be managed without an orthogonal model of persistence and in the ODMG standard, both the C++ and Java models of persistence are not orthogonal.

O_2 originally supported an orthogonal model of persistence, as well as persistence by reachability. However, its C++ binding achieves persistence by inheritance from a root class, called `d_object`, which is obviously not orthogonal persistence. Furthermore, this causes problems for Java, since it implies multiple inheritance, which Java only supports for interfaces. The ODMG standard is also inconsistent in other aspects of its Java binding, since it seems to support persistence by reachability, but the model is not orthogonal persistence.

The final issue of non-traditional object-oriented features dealt with issues such as relationships, name-space management and object identifiers. For example, the ODMG Java binding does not provide support for relationships. Similarly, the O_2 model does not support relationships, but does support relationships in its C++ binding. Also, both the ODMG standard and O_2 use a flat name-space, which is somewhat limiting for multi-user or distributed applications. Finally, in the O_2 C++ binding, object identifiers are directly accessible, which is unsafe. This is caused by pointer mechanisms being available in the language.

To summarise, Suad c ncluded that it was evident that O_2 had influenced the ODMG standard, but there were also many important differences. For example, differences in the type systems, support for parametric classes and support for relationships.

Charting Unexplored Waters: An Adventure in ODB-Land

This paper was written and presented at the workshop by Adam Taylor. The author is with High Road Innovation.

A problem that is sometimes faced by organisations today is whether to use an object-oriented programming language with a relational or object database. Documented examples (e.g. see [Loomi98]), demonstrate the practical benefits of using an object database over a relational database in certain cases. One of the reasons why an organisation may wish to do this is for improving performance where considerable navigational access is required as, for example, in a bill-of-materials or parts-explosion hierarchy. It is precisely this type of problem that Adam described in his presentation.

After initial performance problems experienced with a relational database, Adam's group evaluated several object database products and eventually found ObjectStore to be the best fit for their requirements. However, a number of problems were encountered, which can be summarised as follows:

1. An initial and serious mistake was simply to use the existing relational database model as the basis for the new object database model. This resulted in too many classes and an inflexible design.
2. Casual misuse (e.g. stray deletes) of database objects by developers often caused serious problems.
3. Transforming objects from one form to another was overly-complex, even for simple operations. Despite the transformation being encapsulated, the code was still difficult to write and maintain.
4. Many programming languages were being used in the development group, such as Visual Basic for the GUI and C++ for

database development. Consequently, an in-house language interchange format was written, with developers providing some additional ("glue") code to connect C++ classes and Visual Basic. This interchange format was extended for persistent classes and TCL (used for database loading, unloading, report generation), but resulted in many maintenance problems for the glue code whenever schema changes occurred.

After a period of development, a major reconstruction of classes and schema changes required complete database rebuilds, which reduced database-related errors to almost none. However, further problems subsequently emerged:

1. The schema was overly-complex, which caused schema-evolution problems, particularly in restructuring relationships to meet new requirements. The ObjectStore schema evolution tools were inadequate for this project.
2. The new schema caused synchronisation problems, since not all the developers were working at the same level.

Further restructuring simplified the design, providing easier maintenance, as well as enabling the development of pre-formulated queries. These changes provided significant performance gains.

To conclude, Adam noted that two of the three applications that were being developed on the object database had already been successfully deployed, with the third application due for release by the end of 1997.

Overall, there were many issues related to the use of object databases that were not foreseen. These could be summarised as integration, architectural and deployment strategies.

Object to RDB Mapping: When Attributes and Fields are Not Enough

This paper was written and presented at the workshop by Gerald Zincke. The author is with GMO GmbH, Vienna, Austria.

A common feature of some database systems today is the use of a large centralised mainframe and users "in the field" with small personal computers or laptops (for mobile computing). A typical sequence of events for remote users would be to:

1. Copy portions of the main database to a smaller machine.
2. Make offline changes to data.
3. Write the changed data back to the main database.

However, there are a number of complex issues involved with these requirements:

1. The centralised database has a complex data model. If all the data regarding a small number of business objects are required, this may involve the transfer of a large number of records to the remote offline database.
2. Data manipulation performed on the remote offline database needs to be recorded to enable changes to be correctly applied to the main centralised database.
   - New business objects inserted into the offline database will have to have their keys translated when the data are transferred back to the main database, since there are well-defined rules for key allocation.
   - The same business object may be updated at more than one offline site, so a pre-image log is maintained on each laptop for every updated object.
   - Deleting business objects is also performed by using a log.
   - Transactions must be correctly recorded and applied when data are transferred back to the main database.
3. When transferring the data back, the various changes need to be repeated against the main database. The issues presented above regarding new, updated and deleted objects are carefully applied to ensure that the main database remains consistent.

Gerald described the following general solution to these problems, which was developed using a framework. Business objects:

- Know their primary key and the rules to create that key.
- Know how they are mapped to the database.
- Know pre-image information about themselves.
- Know how to check the database to see whether they already exist.
- Know what relationships they have with other business objects.
- Can only be accessed by a behavioural interface.

So both data and model information are maintained and, using this approach, a solution to the problems outlined earlier was developed in 1.5 months. The framework also allows the same solution to be used for other applications.

Mapping Objects to an RDBMS Using Message Passing

This paper was written and presented at the workshop by Patrick Noonan. The author is with the First Union National Bank, Charlotte, North Carolina.

Patrick described a three-tier system being developed by a large commercial bank. This bank had experienced performance problems with a two-tier solution and was looking towards middleware to provide improved performance for an on-line transaction system that would support 200+ users in 8 states. The servers were large mainframes running Oracle and IMS, with 486 and Pentium PCs being used for the clients. The application uses 110 business objects and 80 Oracle tables.

Early project decisions were to isolate the client and client developers from any knowledge of the database. In this way, they were not influenced by SQL or relational table considerations. Furthermore, business logic would be maintained on the server with thin clients to support data entry and querying using windows and dialog boxes.

Mapping rules for object attributes and classes to row attributes and tables were maintained in an extended data dictionary. This dictionary was used by a custom-built code generator to build server-side code.

Persistent and non-persistent objects were managed by subclassing. Within the development team, the decision to subclass business objects from a persistent superclass caused disagreements, since some developers felt that business objects should not have any knowledge of the persistence layer. However, the solution worked and continued to be used although it was felt that major drawbacks would occur if the relational database was replaced with an object database.

The original performance requirements were that 250,000 bytes from the server should be retrieved in a single transaction within a reasonable (not quantified) time. At the present time, the largest transaction is 80,000 bytes and takes approximately 20 seconds, which is acceptable to customers, since it is relatively infrequent. Other transactions are typically in the 3-5 second range.

To summarise, Patrick felt that the system would continue to use relational databases, although he would prefer to move away from this type of database technology. Object databases were unlikely to be used in the foreseeable future, as the bank was taking a conservative view of this technology.

Conclusions

The previous summaries show that, on the whole, object data management is being successfully used for commercial applications. However, there are still a number of outstanding issues. For example, some vendors have adopted a very aggressive marketing strategy, which means that whilst their products may be used for more applications, there is also the likelihood that there are more failures (which the user community at large generally does not hear about for obvious reasons). Furthermore, from the discussions during the workshop, it also emerged that the participants developing commercial systems were not using the query capabilities provided by a particular object database product. Similarly, none of the same participants had used any public benchmarks for performance evaluations; most preferred to undertake their own benchmarking tests. On the issue of standards, there are some problems with the current efforts and a number of areas that ODMG needs to address were clearly identified. Another important issue was that, for successful commercial systems, it is essential to have suitably qualified people. Retaining such people can also be a major problem.

Workshop participants may make available electronic copies of their position papers and slides. If so, links to these will be added from one of the workshop home pages located at:

`http://www.soi.city.ac.uk/~akmal/oopsla97.dir/workshop.html`.

Acknowledgements

The workshop organisers would like to thank all those who took the time and effort to prepare position papers, presentations or participated directly in the workshop. Thanks to Doug Barry for his comments on an earlier draft of this report.

References

[Loomi98] M.E.S. Loomis and A.B. Chaudhri (eds.) (1998) Object databases in practice (Upper Saddle River, New Jersey: Prentice-Hall) `http://www.prenhall.com/ptrbooks/ptr_013899725x.html`

[Zorn95] B.G. Zorn and A.B. Chaudhri (1995) Object database behavior, benchmarks, and performance. *OOPS Messenger*. 6 (4):159-163. `http://www.cs.colorado.edu/homes/zorn/public_html/oopsla95/addendum.ps`

Citation

International Conference on Management of Data and Symposium on Principles of Database Systems >archive
Proceedings of the 1999 ACM SIGMOD international conference on Management of data >toc
1999 , Philadelphia, Pennsylvania, United States

An XJML-based wrapper generator for Web information extraction

> Also published in ...

Authors
Ling Liu
Wei Han
David Buttler
Calton Pu
Wei Tang

Sponsors
SIGART : ACM Special Interest Group on Artificial Intelligence
SIGMOD : ACM Special Interest Group on Management of Data
SIGACT : ACM Special Interest Group on Algorithms and Computation Theory

Publisher
ACM Press New York, NY, USA

Pages: 540 - 543 Series-Proceeding-Article
Year of Publication: 1999
ISSN:0163-5808

doi> http://doi.acm.org/10.1145/304182.304570 (Use this link to Bookmark this page)

> full text > references > citings > index terms > peer to peer

> Discuss > Similar > Review this Article Save to Binder

> BibTex Format

↑ FULL TEXT: Access Rules

pdf 512 KB

↑ REFERENCES

Note: OCR errors may be found in this Reference List extracted from the full text article. ACM has opted to expose the complete List rather than only correct and linked references.

1 A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A Query Language for XML. http://www.w3c.org/TR/1998/NOTE-xml-ql- 19980819, 1998.

2 Joachim Hammer , Héctor García-Molina , Svetlozar Nestorov , Ramana Yerneni , Marcus Breunig , Vasilis Vassalos, Template-based wrappers in the TSIMMIS system, Proceedings of the 1997 ACM SIGMOD international conference on Management of data, p.532-535, May 11-15, 1997, Tucson, Arizona, United States

3 Craig A. Knoblock , Steven Minton , Jose Luis Ambite , Naveen Ashish , Pragnesh Jay Modi , Ion Muslea , Andrew G. Philpot , Sheila Tejada, Modeling Web sources for information integration, Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence, p.211-218, July 1998, Madison, Wisconsin, United States

4 N. Kushmerick, D. Weil, and R. Doorenbos. Wrapper induction for information extraction. In Proceedings of Int. Joint Conference on Artifical Intelligence (IJCAI}, 1997.

5 L. Liu, C. Pu, and W. Tang. Continual queries for internet-scale event-driven information delivery. IEEE Knowledge and Data Engineering, 1999. Special Issue on Web Technology.

6 L. Liu, C. Pu, W. Tang, and W. Han. CONQUER: A Continual Query System for Update Monitoring on the WWW. international Journal of Computer Systems, Science and Engineering, 1999. Special Issue on WWW Semantics, edited by Dan Suciu and Letizia Tanca.

7 Ling Liu , Calton Pu , Wei Tang , David Buttler , John Biggs , Tong Zhou , Paul Benninghoff , Wei Han , Fenghua Yu, CQ: a personalized update monitoring toolkit, Proceedings of the 1998 ACM SIGMOD international conference on Management of data, p.547-549, June 01-04, 1998, Seattle, Washington, United States

↑ CITINGS *3*

Elke A. Rundensteiner , Andreas Koeller , Xin Zhang, Maintaining data warehouses over changing information sources, Communications of the ACM, v.43 n.6, p.57-62, June 2000

David Mattox , Len Seligman , Ken Smith, Rapper: a wrapper generator with linguistic knowledge, Proceedings of the second international workshop on Web information and data management, p.6-11, November 02-06, 1999, Kansas City, Missouri, United States

Yue-Shan Chang , Min-Huang Ho , Wen-Chen Sun , Shyan-Ming Yuan, Supporting unified interface to wrapper generator in integrated information retrieval, Computer Standards & Interfaces, v.24 n.4, p.291-309, September 2002

↑ INDEX TERMS

Primary Classification:

I. Computing Methodologies

↳ **I.7** DOCUMENT AND TEXT PROCESSING

↳ **I.7.2** Document Preparation

↳ **Nouns:** XML

Additional Classification:

H. Information Systems

↳ **H.3** INFORMATION STORAGE AND RETRIEVAL

↳ **H.3.3** Information Search and Retrieval

↳ **Subjects:** Information filtering

↳ **H.3.5** On-line Information Services

↳ **Subjects:** Web-based services

↳ **H.5** INFORMATION INTERFACES AND PRESENTATION (I.7)

↳ **H.5.3** Group and Organization Interfaces

↳ **Subjects:** Web-based interaction

General Terms:
Design, Languages, Management, Performance, Theory

↑ Peer to Peer - Readers of this Article have also read:

- Data structures for quadtree approximation and compression
  Communications of the ACM 28, 9
  Hanan Samet

- The state of the art in automating usability evaluation of user interfaces
  ACM Computing Surveys (CSUR) 33, 4

- A lifecycle process for the effective reuse of commercial off-the-shelf (COTS) software
  Proceedings of the 1999 symposium on Software reusability
  Christine L. Braun

- A catalog of techniques for resolving packaging mismatch
  Proceedings of the 1999 symposium on Software reusability
  Robert DeLine

- Using adapters to reduce interaction complexity in reusable component-based software development
  Proceedings of the 1999 symposium on Software reusability
  David Rine , Nader Nada , Khaled Jaber

↑ This Article has also been published in:

- ACM SIGMOD Record
  Volume 28 , Issue 2 (June 1999)

An XML-based Wrapper Generator for Web Information Extraction

Ling Liu, Wei Han, David Buttler, Calton Pu, Wei Tang

Oregon Graduate Institute of Science and Technology
Department of Computer Science and Engineering
P.O.Box 91000 Portland, Oregon 97291-1000 USA
{lingliu,weihan,buttler,calton,wtang}@cse.ogi.edu

1 Introduction

There has been tremendous interest in information integration systems that automatically gather, manipulate, and integrate data from multiple information sources on a user's behalf. Unfortunately, web sites are primarily designed for human browsing rather than for use by a computer program. Mechanically extracting their content is in general a rather difficult job if not impossible [4]. Software systems using such web information sources typically use hand-coded *wrappers* to extract information content of interest from web sources and translate query responses to a more structured format (e.g., relational form) before unifying them into an integrated answer to a user's query. The most recent generation of information mediator systems (e.g., Ariadne [3], CQ [5, 7], Internet Softbots [4], TSIMMIS [2]) addresses this problem by enabling a pre-wrapped set of web sources to be accessed via database-like queries.

However, hand-coding a wrapper is time consuming and error-prone. We have also observed that, by using a good design methodology, only a relatively small part of the code deals with the source-specific access details, the rest of the code is either common among wrappers or can be expressed in a high level, more structured fashion. As the Web grows, maintaining a reasonable number of wrappers becomes impractical. First, the number of information sources of interest to a user query can be quite large, even within a particular domain. Second, new information sources are constantly added on the Web. Thirdly, the content and presentation format of the existing information sources may change frequently and autonomously.

With these observations in mind, we have developed a wrapper generation system, called XWrap, for semi-automatic construction of wrappers for Web information sources. The system contains a library of commonly used functions, such as receiving queries from applications, handling of filter queries, and packaging results. It also contains some source-specific facilities that are in charge of mapping a mediator query to a remote connection call to fetch the relevant pages and translating the retrieved page(s) into a more structured format (such as XML documents or relational tables). A distinct feature of our wrapper generator is its ability to provide an XML-enabled, feedback-based, interactive wrapper construction facility for Internet information sources. By XML-enabled we mean that the extraction of information content from the Web pages will be captured in XML form and the process of filter queries is performed against XML documents. By feedback-based we mean that the wrapper construction process will be revisited and tuned according to the feedback received by the wrapper manager.

SIGMOD '99 Philadelphia PA

2 Methodology

The philosophy behind our "XML-enabled" wrapper generation methodology is to develop mechanisms that provides a clean separation of the semantic knowledge of information extraction from the wrapper code generation using a rule-based approach. More concretely, the wrapper generator first exploits formatting information in Web pages to hypothesize the underlying semantic structure of a page, and then encode the hypothetical structure and the information extraction knowledge of the web pages in a rule-based declarative language designed specifically for XWrap information extraction. From the set of information extraction rules and the XML-templates derived throughout the XWrap walkthrough sessions, the system constructs a wrapper program that facilitates both the tasks of querying of a semi-structured Web source and integrating it with other Web information sources. We partition the wrapper generation process into a series of sub-processes called *phases*, as shown in Figure 1. A phase is a logically cohesive operation that takes as input one representation of the source document and produces as output another representation.

The first phase is called the *structure analysis*. It performs the encoding of information extraction knowledge into a set of implementation-independent information extraction rules. It involves three tasks: (1) fetching a Web page from a remote site and generating a tree-like structure for the page, (2) identifying regions and tokens that are key components or key element templates for extracting information content from the page, and (3) inferring the nesting hierarchy of sections, which represents the syntactic structure of the page. For a Web page in HTML, this phase builds a source-specific HTML parser that wraps the page into a tree structure, annotates content tokens in comma-delimited format and nesting hierarchy in an XML-template file (see Section 3 for an example).

The second phase is called the *source-specific XML generation*. It involves two steps: (1) Generate a source-specific XML template file based on the content tokens and the nesting hierarchy specification obtained in the structure analysis

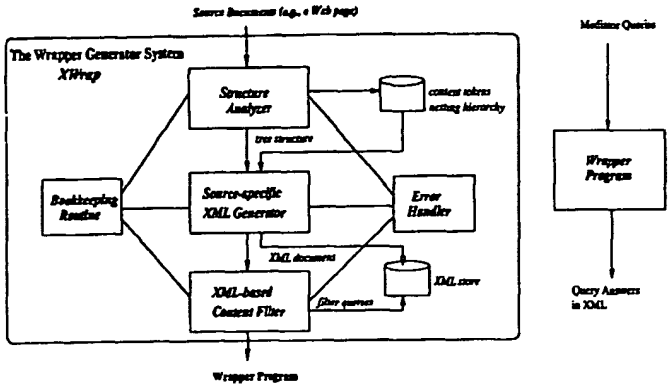

Figure 1: Phases of an XML-enabled Wrapper Generator

phase; and (2) Construct a source-specific XML generator using the template-based XML generator. An XML template is a well-formed XML document. In addition to constructs that one would normally expect in an XML file, it also contains a small set of processing instructions and special placeholders defined by the XML generator.

The third phase is called the *generic XML-based content filter (content extraction controller)*. After wrapping a Web source in an XML format, we build a generic XML-based content filter that is capable of handling complex conjunctive or disjunctive queries handed over by various mediator applications.

The *bookkeeping* routine of the wrapper collects information about all the data objects that appear in the source document, keeps track of the names used by the program, and records essential information about each. For example, a wrapper needs to know how many arguments a tag expects, whether an element represents a string or an integer.

The *error handler* is designed for the detection and reporting errors in the source document. It is invoked when a flaw in the source document is detected. It must warn the wrapper developer by issuing a diagnostic, and adjust the information being passed from phase to phase so that each phase can proceed. It is desirable that wrapper construction be completed on flawed source documents, at least through the structure-analysis phase, so that as many errors as possible can be detected in one construction pass. The error messages should allow the wrapper developer to determine exactly where the errors have occurred. Both the bookkeeping and error handling routines interact with all phases of the wrapper generator.

3 A Walk Through Example

3.1 Information Extraction

Consider the weather report page for Savannah, GA at the national weather service site, and a fragment of HTML document for this page in Figure 2.

Figure 3 shows a portion of the HTML tree structure, corresponding to the above HTML fragment, which is generated by running a source-specific parser on the Savannah weather source page. In this portion of the HTML tree, we have the following six types of tag nodes: `TABLE`, `TR`, `TD`, `B`, `H3`, `FONT`, and a number of semantic token nodes at leaf node level, such as `Maximum Tempature`, `Minimum Tempature`, `84.9(29.4)`, `64.0(17.8)`, etc.

In the current implementation, the structure analyzer sets up three panels in an information extraction window. The two top level panels displays the web page and its HTML tree structure, and the bottom panel is used to display the information extraction rules and the comma delimited file generated.

Based on the interaction with the wrapper developer, the region extractor identifies four semantic regions of interest for extraction, each is a table in the nws.noaa.gov web page. The semantic-token extractor infers that the leaf node `Maximum` and `Minimum Temperatures` is the heading of a table section, the string `Maximum Tempature F(C)` is a semantic token with the string itself as the token name and the string `84.9 (29.4)` as the token value, and so forth. By traversing the entire tree structure, the semantic-token extractor produces as output, a set of information extraction rules for extracting the semantic tokens into a comma-delimited file and the hierarchical structure into an XML-tamplate file for each of the nws.noaa.gov current weather report page. Consider the fragment of the tree structure given in Figure 3. By walking through the left most branch of the tree with the user's feedback, XWrap can infer that `Maximum` and `Minimum Temperatures` is the table name because it is in between a pair of header tags `<H3>` and `</H3>`. Also based on the observation that all the rest of children nodes of `TABLE` are of the same type `TR`, and each again has three children nodes. The leaf node `Maximum` and `Minimum Temperatures` of the first branch of `TABLE` is marked as the table name, the leaf nodes `Maximum Temperature`, `Minimum Temperature` of the second branch of `TABLE` as column names, and each of the rest of the branches of `TABLE` as an instance of the second `TR` node type. We develop an algorithm that, given a page with all regions (tables and text sections, and headings identified, outputs an XML template for the page. Figure 4 shows the fragment of the XML template file corresponding to the part of a NWS weather report page shown in Figure 3.

Due to the fact that the heuristics used for identifying sections and headings may have exceptions for some information sources, it is possible for the system to make mistakes when trying to identify the hierarchical structure of a new page. For example, based on the heuristic on font size, the system may identify some words or phrases as headings when they are not, or fail to identify phrases that are headings, but do not conform to any of the pre-defined regular expressions. We have provided a facility for the user to interactively correct the system's guesses. Through a graphical interface the user can highlight tokens that the system misses, or delete tokens that the system chooses erroneously. Similarly, the user can correct errors in the system-generated XML-template that describes the structure of the page.

3.2 Source-specific XML generator

The source-specific XML generator consists of an XML Template Engine and an XML parser. The XML template engine generates XML statements using both the comma-delimited file and the XML template parsed by the XML parser. Comparing with the normal XML documents, XML templates are well-formed XML files that contain processing instructions. Such instructions are used to direct the template engine to the special placeholders where data fields should be inserted into the template. For instance, the processing instruction **XG-InsertField-XG** has the canonical form of

```
<TABLE><TR><TD COLSPAN=3><H3><FONT FACE="Arial, Helvetica">Maximum and Minimum Temperatures</FONT>
</H3> </TD></TR><TR><TD ALIGN=CENTER BGCOLOR="#FFFFFF"><B><FONT COLOR="#0000A0"><FONT FACE=
"Arial,Helvetica">Maximum<BR>Temperature<BR>F(C)</FONT></FONT></B></TD><TD ALIGN=CENTER BGCOLOR=
"#FFFFFF"><B><FONT COLOR="#0000A0"><FONT FACE="Arial,Helvetica">Minimum<BR>Temperature<BR>F(C)
</FONT></FONT></B></TD><TD></TD></TR><TR><TD ALIGN=CENTER><FONT FACE="Arial,Helvetica">82.0(27.8)
</FONT></TD><TD ALIGN=CENTER><FONT FACE="Arial, Helvetica">62.1(16.7)</FONT></TD><TD><FONT FACE=
"Arial, Helvetica">In the <B>6 hours</B> preceding Oct 29, 1998 - 06:53 PM EST / 1998.10.29 2353
UTC</FONT></TD></TR><TR><TD ALIGN=CENTER><FONT FACE="Arial,Helvetica">80.1(26.7)</FONT></TD>
<TD ALIGN=CENTER><FONT FACE="Arial, Helvetica">45.0(7.2)</FONT></TD><TD><FONT FACE="Arial,
Helvetica">In the <B>24 hours</B> preceding Oct 28, 1998 - 11:53 PM EST / 1998.10.28 0453 UTC</FONT>
</TD></TR><TR><TD COLSPAN=3><HR SIZE=1 NOSHADE WIDTH="100%"></TD></TR></TABLE> .....
```

Figure 2: An HTML fragment of the weather report page at nws.noaa.gov site

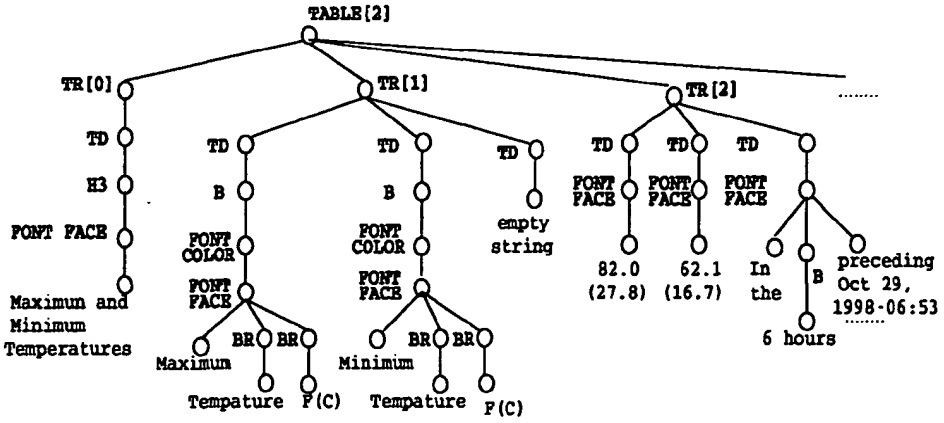

Figure 3: Another fragment of the HTML tree for the Savannah weather report page

<?XG-InsertField-XG ''FieldName"?>. It looks for a field with a specified name "FieldName" in the comma-delimited file and inserts that data at the point of the processing instruction.

An XML template also contains a repetitive part. The XG-Iteration-XG processing instruction determines the beginning and the end of a repetitive part. A repetition can be seen as a loop in classical programming languages. After the template engine reaches the "End" position in a repetition, it takes a new record from the delimited file and goes back to the "Start" position to create the same set of XML tags as in the previous pass. New data is inserted into the resulting XML file.

3.3 XML-based Content Filter

The XML-based Content Filter is the interface between the mediator application and the wrapper. The main tasks of the content filter include

- Accept a mediator query, identify network locations of the pages needed to answer the query, and call the data wrapper manager to wrap the source page(s) into the XML format;
- Translate the complex content-sensitive mediator queries over the given Web pages to SQL-like XML queries.

For single-document sources, this is straightforward, i.e., the URL for that page is known to the wrapper. For sources with multiple documents of the same type, a mapping between a query and the URL of the relevant page may be required. To provide the capability of determining the network location of the page relevant to a query, the wrapper developer specifies a mapping function which takes necessary arguments from a query (e.g., the city name and its four-character code on the NWS weather source) and constructs a URL pointing to the page to be fetched. For instance, in the NWS weather report site there is a one to one mapping between the city name and the URL of the page for that city. This mapping can be obtained by querying the city from either the US weather report home page or the world weather report home page. Currently we use Java programs for the purpose of making HTTP connections to the Web information sources and retrieving data from them.

To support information filtering over XML documents, we would need to understand the data structure used to hold the XML documents and the types of query operators a system would like to support. One popular approach is to use XML-QL style of query languages [1]. The main problems with this proposal is the availability of the software to support such an XML-QL based system. Another approach is to transform an XML document into a collection of relational tables or a class of complex objects. There are two main challenges in transforming an XML document to a collection of relational tables (or object classes). First, given an XML document, we need to decide how many relational tables are needed to capture the nesting structure of the XML file. Second, for multi-document sources, the XML

```
.....
<Maximum_and_Minumum_Temperatures>
<Description>Maximum and Minimum Temperatures</Description>
<!-- Start of the repetition -->
<?XG-Iteration-XG ''Start"?>
  <Maximum_and_Minimum_Temperatures_Child>
    <Maximum_Temperature>
    <Description>MaximumTemperature F(C)</Description>
    <Value><?XG-InsertField-XG ''Maximum Tempature"></Value>
    </Maximum_Temperature>

    <Minimum_Temperature>
    <Description>MinimumTemperature F(C)</Description>
    <Value><?XG-InsertField-XG ''Minimum Temperature"></Value>
    </Minimum_Temperature>

    <TD>
    <Description></Description>
    <Value><?XG-InsertField-XG ''TD"></Value>
    </TD
  </Maximum_and_Minumum_Temperatures_Child>
<?XG-Iteration-XG ''End"?>
<!-- End of the repetition -->
</Maximum_and_Minumum_Temperatures>
.....
```

Figure 4: An Example XML template for a portion of the NWS current weather report page

document generated at Phase two is only an instance of the corresponding source. Quite often it may have some missing sections, or missing columns of a table section. Thus, the relational tables that capture the given XML document may not be generic enough to capture other instance pages of the same source. For example, consider the NWS current weather report source. For some cities in US, the weather report at a given time point may include the `Precipitation Accumulation` section, although at other times this section is not available. Another example is the case when some column or sub-section (such as the `Dew Point` field) is missing in one document instance but appeared in another instance document. Therefore, instead of using a pre-defined number of relational tables and pre-defined table formats, the design of the content filter should determine the number of relational tables to use and the table formats at run-time.

4 Description of Demo

We demonstrate the latest version of our wrapper toolkit as described in the previous sections. Specifically we show how to use our wrapper generator toolkit to construct wrappers for the following three classes of Internet information sources.

1. Web sources that are multiple-instance sources (i.e., multiple documents of the same type), and organize their content information in multiple two-dimensional tables, such as the current weather report source at the national weather service site, the stock quote information source at stockmaster.com or Yahoo investment site.

2. Web sources that are multiple-instance sources, and organize the content information in multiple nesting sections, such as the CIA factbook web site or the Sun Site Web Museum site.

3. Web sources that are single-instance sources, such as SIGMOD'99, ICDE'99 or VLDB'99 conference web sites.

Although all three classes of Web sources support different search and access methods, the wrappers generated hide all source specific details from the applications and end-users by providing a common interface to the underlying data independently of where and how it is stored. Furthermore, we demonstrate the filter query processing capabilities of our wrappers.

As part of the Continual Queries project, we have also developed a graphical browsing tool that lets users submit queries to wrappers, navigate through the XML files generated for the Web pages that have been wrapped, and zoom in on the CQ answer objects and their nested hierarchies as necessary.

Acknowledgement

This research is partially supported by DARPA contract MDA972-97-1-0016, Intel, and Boeing. Our thanks are also due to the Master students involved in the CQ and XWrap projects at OGI, especially Divya Kumar, Shujing Liu, Jaya Shrivastav, and Iffath Zofishan.

References

[1] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A Query Language for XML. *http://www.w3c.org/TR/1998/NOTE-xml-ql-19980819*, 1998.

[2] J. Hammer, M. Brennig, H. Garcia-Molina, S. Nesterov, V. Vassalos, and R. Yerneni. Template-based wrappers in the tsimmis system. In *Proceedings of ACM SIGMOD Conference*, 1997.

[3] C. A. Knoblock, S. Minton, J. L. Ambite, N. Ashish, P. J. Modi, I. Muslea, A. Philpot, and S. Tejada. Modeling web sources for information integration. In *Proceedings of AAAI Conference*, 1998.

[4] N. Kushmerick, D. Weil, and R. Doorenbos. Wrapper induction for information extraction. In *Proceedings of Int. Joint Conference on Artifical Intelligence (IJCAI)*, 1997.

[5] L. Liu, C. Pu, and W. Tang. Continual queries for internet-scale event-driven information delivery. *IEEE Knowledge and Data Engineering*, 1999. Special Issue on Web Technology.

[6] L. Liu, C. Pu, W. Tang, and W. Han. CONQUER: A Continual Query System for Update Monitoring on the WWW. *International Journal of Computer Systems, Science and Engineering*, 1999. Special Issue on WWW Semantics, edited by Dan Suciu and Letizia Tanca.

[7] L. Liu, C. Pu, W. Tang, J. Biggs, D. Buttler, W. Han, P. Benninghoff, and Fenghua. CQ: A Personalized Update Monitoring Toolkit. In *Proceedings of ACM SIGMOD Conference*, 1998.